`INSERT from Reality`: A Schema-driven Approach to Image Capture of Structured Information

Undergraduate Honors Research Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Bachelor of Science in Computer Science and Engineering with Honors Research Distinction in the College of Engineering at The Ohio State University

By

Mohit Deshpande

Undergraduate Program in Computer Science and Engineering

The Ohio State University

2018

Examination Committee:

Dr. Arnab Nandi, Advisor

Dr. Srinivasan Parthasarathy

Dr. Jeremy Morris

Abstract

There is a tremendous amount of structured information locked away on document images, e.g., receipts, invoices, medical testing documents, and banking statements. However, the document images that retain this structured information are often ad hoc and vary between businesses, organizations, or time periods. Although optical character recognition allows us to digitize document images into sequences of words, there still does not exist a means to identify schema attributes in the words of these ad hoc images and extract them into a database. In this thesis, we push beyond optical character recognition: while current information extraction techniques use only optical character recognition from structured images, we infer the visual structure and combine it with the textual information on the document image to create a highly-structured `INSERT` statement, ready to be executed against a database. We call this approach Ifr. We use OCR to obtain the textual contents of the image. Our natural language processes annotate this with relevant information such as data type. We also prune irrelevant words to improve performance in subsequent steps. In parallel to textual analysis, we visually segment the input document image, with no a-priori information, to create a visual context window around each textual token. We merge the two analyses to augment the textual information with context from the visual context windows. Using analyst-defined heuristic functions, we can score each of these context-enabled entities to probabilistically construct the final `INSERT` statement. We

evaluated IFR on three real-world datasets and were able to achieve F_1 scores of over 83% in `INSERT` generation on these datasets, spending approximately 2 seconds per image on average. Comparing IFR to natural language processing approaches, such as regular expressions and conditional random fields, we found IFR to perform better at detecting the correct schema attributes. To compare IFR to a human baseline, we conducted a user study to find the human baseline of `INSERT` quality on our datasets and found IFR to produce `INSERT` statements that were comparable or exceeded that baseline.

Acknowledgments

First and foremost, I would like to thank Arnab for his patience and understanding while I was getting accustomed to the reigns of undergraduate research. I greatly appreciate the guidance Arnab provided when I was stuck with a problem such as framing an introduction, future work, or general paper writing.

I'd also like to express my gratitude to Behrooz. During his post-doc at the university, I learned most of what I know about paper and scientific writing from him. He always had the time to explain how he wrote a paper section and help me debug code.

Finally, I would like to thank the data team organization's PhD students: Protiva, Meraj, Niranjan, and Ritesh for helping me with paper writing, bug finding/fixing, and developing analytical skills. They have been of great help to me and my work from the very start.

This material is based upon work supported by the National Science Foundation under REU career grant no. 1453582.

Vita

May 2014	Beavercreek High School
August 2014 – present	The Ohio State University
Summer 2014 – present	Instructor, Zenva
Summer 2015 – present	Undergraduate Researcher, The Ohio State University
Summer 2014 and 2015	Undergraduate Research Intern, Air Force Research Lab
Summer 2016	Software Engineering Intern, Hyland Software Inc.
Summer 2017 and 2018	Software Engineering Intern, Amazon Lab 126

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

Page

List of Tables

Table **Page**

List of Figures

Chapter 1: Introduction

The combination of smartphone cameras and high-quality, commoditized computer vision services, such as Google Cloud Vision[1] and Microsoft Cognitive Services[2], has allowed us to gain a richer understanding of our world. In fact, over one trillion images were taken in 2015 at growth rate of 16.2% [40]. One particularly interesting and useful subset of captured images are *document images* such as receipts, flyers, invoices, or any image with text, as shown in Figure 1.1. Locked away on these images are data that would be more useful in a structured digital medium. After which, they may be used for data analysis processes such as banking [14], health data analysis [13], and traffic monitoring [9].

As an example of the critical information locked away in document images, consider electronic health records and the database management systems (DBMSes) that back them. Although there exists a suggested centralized format, each hospital may use its own. When a patient travels to another hospital, the attending physician requires the patient's medical history. This history is in the form of scans or images which the physician must parse through whenever she needs information. Furthermore, the document images of the patient's medical history may not be in a format

[1] `https://cloud.google.com/vision/`

[2] `https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/`

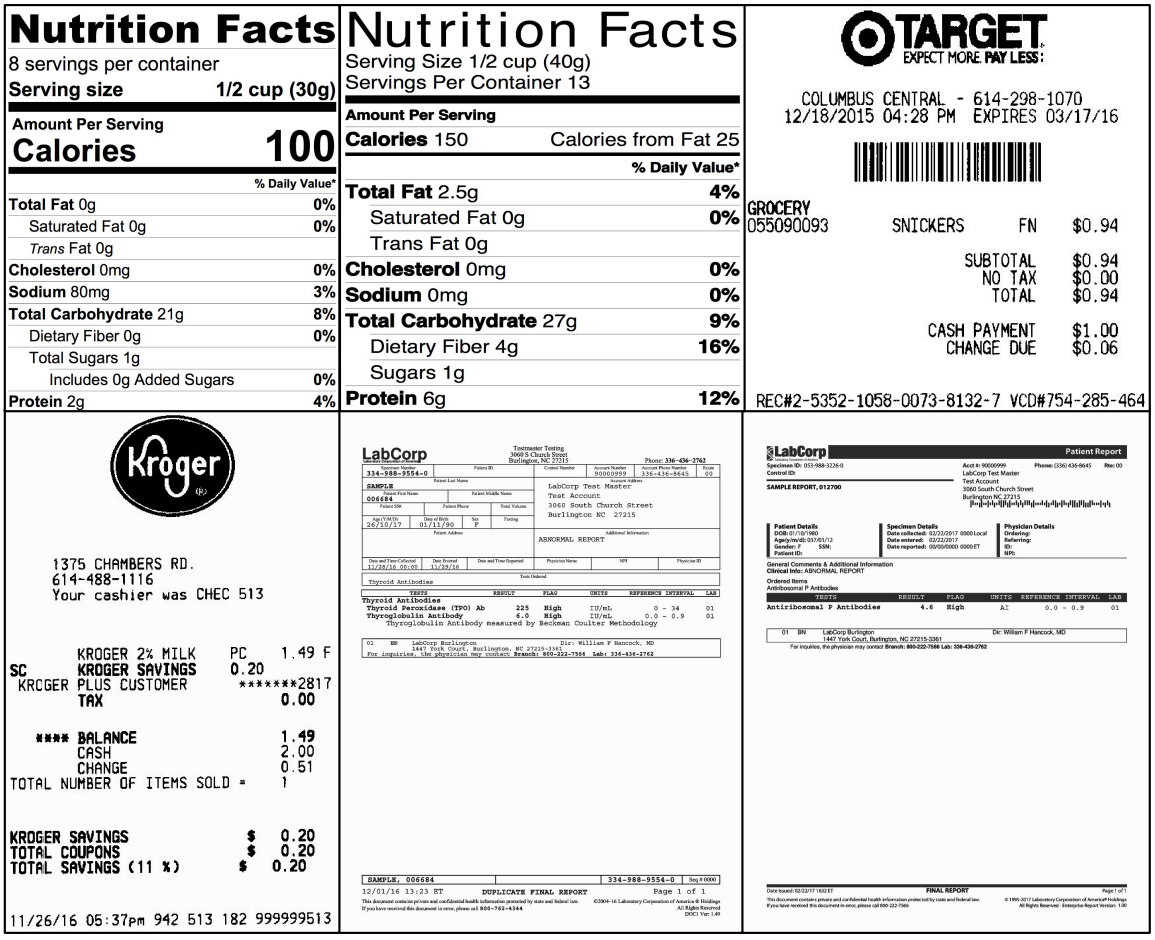

Figure 1.1: Examples of document images. These are taken from our three evaluation datasets: nutrition facts, receipts, and Labcorp medical testing forms.

familiar to the physician. This illustrates the gap between unstructured images and the highly-structured DBMS. The better solution to this problem is to extract the relevant information from the document images of the patient's medical history into the new hospital's DBMS, making all prior medical history instantly queryable.

Previous work for information extraction from highly-structured and organized images has existed for decades. For example, most banking applications allow users to upload an image of their check to deposit it (after being verified by a human). As another example, Scantron machines used in classrooms use image processing to detect which answer the student selected. In these cases, the input images are highly-structured, and the algorithms operating on them have complete knowledge of the input.

However, in real-world scenarios, relevant information may be present on document images with varying visual formats. In this case, prior techniques fail because of the varying visual structure. With this variance, we cannot make any a-priori assumptions about the input document image. However, all of the information we need to extract is in the text on that document image, regardless of visual structure. Hence, there is a need for a domain-independent solution to information extraction from these document images.

Optical character recognition (OCR) is an integral part of digitizing these document images as it produces the bag-of-words. However, not only is the bag-of-words unordered and may contain errors, but it also completely throws away the context and structure of the document image by reducing it to a sequence of unordered unigrams. The context around these words, taken from the visual structure of the document image, is essential to extracting the information on that document image.

In this thesis, we present a novel approach called IFR, which allows for the domain-independent extraction of information from images with any visual structure. Using IFR, we can bridge ad hoc document images and highly-structured data analysis.

1.1 Challenges

We enumerate some challenges with using unstructured documents for extracting attribute information.

No structural indicators. In a raw image, we only have pixels to guide our layout analysis. Other work [4, 8] uses HTML or PDF structural information, e.g., `<table>` or PDF table meta-structures, to identify organization. We do not have these meta-data in a document image and must use the raw pixels to infer a layout.

OCR character-level errors. OCR often makes character-level mistakes that hinder high-quality information extraction. For example, if we were looking for a cost value near the word "balance" and OCR makes a mistake, e.g., "bolance", we may not find the cost value. A common character mistake is swapping a comma for a period, thus making it more difficult for regular expression parsers to extract information. The naïve solution is to enumerate all possible homoglyph transcriptions which is an combinatorially large problem.

Combinatorially large candidate space. Given a bag-of-words, each attribute may be assigned to any subset of consecutive words, i.e., n-grams. These n-grams may be unigrams or larger, up to the size of the bag-of-words in the document image. Checking all possible n-grams for a single attribute match is a combinatorially large problem.

1.2 Motiving Examples & Use-cases

One helpful use-case for Ifr is deriving insights in augmented reality (AR) systems. The final output of Ifr, once `INSERT`ed into a database, can be used to help the user discover insights in the real world. Consider the example of generating AR charts and graphs from data in a database: a user is trying to track her spending and wants to generate graphs that display where money was spent each month. When receiving a new paper receipt, an AR system, such as Microsoft Hololens or Google Glass, can display an AR pie or bar chart that shows new spending information, given the new paper receipt. To create these visualizations, we require Ifr to extract information from this paper receipt and `INSERT` it into the user's spending database (or perhaps link with a third-party money tracking service).

Another use-case is regarding the aforementioned electronic health record scenario: integrating and unifying different patient history records. When a physician opens up a patient chart, the AR system can overlay and highlight sections of the patient's history. However, we require Ifr to digitize and integrate these into the patient electronic health record.

1.3 Contributions

We have developed Ifr, a means to extract schema attributes from unstructured images with no a-priori information. Overall, the contributions of this thesis are as follows.

- We introduce a variant of bottom-up segmentation specific to analyzing layout in unstructured document images (Section 3.5).

- We describe a novel approach to combining visual, textual, and schema-level information to identify candidates for `INSERT`. (Section 3.4–3.6).
- We also describe an approach to scoring these candidates to create an attribute assignment (Section 3.7–3.8).

Chapter 2: Related Works

IFR is a multi-faceted approach to extracting `INSERT` statements from ad hoc, unstructured images. IFR combines work from computer vision, natural language processing, and databases.

2.1 Structure Discovery

Learning the structure of images and tables has been studied [4, 8, 32, 37]. However, these use some kind of meta-structure such as HTML tags or PDF data [4, 8, 27, 37, 36]. Specifically, [4, 37] uses HTML tables while [36] uses PDF data to extract information. However, IFR has none of these structures since it operates on raw images, making our problem more challenging. We use a variant of bottom-up segmentation to generate this layout, similar to [17, 33].

2.2 DeepDive

A similar work to ours is DeepDive [32]. IFR considers unstructured input while DeepDive uses mainly digitized text data. DeepDive simply runs an image through OCR. This extra step of digitization presents more challenges for IFR, particularly

with missing and incorrect text. DeepDive's user-defined functions (UDFs) are analogous to IFR's heuristic functions. However, DeepDive's UDFs are meant to be high-recall and low-precision while IFR's heuristics are both high-recall and high-precision. Hence, IFR provides more utilities and tools, such as homoglyph-invariance and edit distance functions, for the analyst to use when creating these functions. Additionally, IFR does not require any training data to operate: the analyst simply needs to change the heuristic functions. In both DeepDive and IFR, these heuristic functions are not replaced by machine learning because they are simpler for analysts to understand, write, and use.

2.3 Schema Matching

We can rephrase the problem of IFR as a schema matching problem where the source is a document image and the target is a schema. There exists much previous work on schema matching. The first step of state-of-the-art approaches is to find points of similarity between the source and target. These approaches assume the source and target are of the same type. However, for IFR, there are no points of commonality since the source are target are completely different structures: a document image and a schema, respectively.

Conditional Random Fields (CRFs) [28, 23] can be utilized for schema matching and information extraction from textual data [39]. Linguistic matching [27] uses names and descriptions for creating the schema matching. Machine learning techniques [30, 10] use meta-data and statistics learned from the tuples in the schema. Graph and structure-based techniques [11, 29] uses clusters in the source and target for schema matching. Constraint-based schema matching techniques [1] use data

types, primary and foreign keys, and data ranges. IFR is more in-line with rule-based [31] and self-tuning [24, 26] schema matching. The heuristics for IFR can be viewed as iterative rules that adjust the result of the resulting `INSERT` statement.

Using the aforementioned schema matching approaches, we cannot directly `INSERT` document images. The first reason is that there is no similarity between the source, a document image, and target, a schema. The second reason is that the output of IFR produces an `INSERT` statement from the document image, as opposed to creating a set of matches such as SLT or XQuery [2]).

2.4 Digitization

A critical use-case for IFR is the rapid and autonomous digitization of document images into a database, instantly making the content queryable. CAPFF [38] is a related digitization work that is involved much earlier in the process: when the user is filling out the form. CAPFF uses a fixed document style and visual markers to provide context and aid users in completing documents. IFR does not operate on handwritten text and is focused towards document images created by a machine rather than a human. Additionally, IFR does not require a fixed document structure; CAPFF and IFR both use visual markers, but the visual markers used by IFR are inferred from the document image itself. CAPFF is a semi-supervised approach that requires a human-in-the-loop for each step, but IFR's heuristic functions are defined by the analyst only once and can be changed later at will.

Other paper tools, such as Usher [6] and printr [5], have also considered document images. Specifically, Shreddr [7] also considers document images and makes liberal

use of document image templates. We show that using templates produces poor precision and recall values when the visual structure of the document images vary.

Chapter 3: INSERT From Reality

In this chapter, we describe the implementation details of IFR. Section 3.1 enumerates all of the notation that will be used for the rest of this thesis. Section 3.2 gives a brief, high-level overview of the different subprocesses in IFR. Section 3.3 describes document and schema classification. Section 3.4 discusses the text analysis steps, and Section 3.5 describes the parallel image analysis. Section 3.6 illustrates how schema matching candidates are generated, and Section 3.7 describes how our analyst-defined heuristic functions are used to score these candidates. With these scores, Section 3.8 describes how to build the final `INSERT` statement. Finally, Section 3.9 discusses some of the limitations of IFR.

3.1 Background

Before discussing the implementation of our approach, we must first formally define all of the entities used as input and in the implementation.

We model our input binary image as an $m \times n$ matrix I where $I[i, j]$ represents the pixel in the i^{th} row and j^{th} column. Each input image may have a different size, but images are processed independently. Using conventional computer vision notation, each pixel in this binary image is either white ($I[i, j] = 1$) or black ($I[i, j] = 0$). Also, the origin is the first pixel at the top-left of the image. The number of black

pixels in a row r or column c of image I is useful to know and can be determined using $\sum_{j=1}^{n} \delta(I[r, j], 0)$ and $\sum_{i=1}^{m} \delta(I[i, c], 0)$ for rows and columns, respectively. We define $\delta(i, j)$ to be the Kronecker delta in function notation.

$$\delta(i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

For our textual data, we model the output of OCR as a bag-of-words $W = \{w_k\}$. Each word is enriched with the bounding box coordinates in the image, with respect to the conventional image coordinate system.

Given a relational database, we define a set of schema $\{S_\ell\}$ where attribute $a_c^{(\ell)}$ is the cth attribute of schema ℓ. Each attribute has some metadata associated with it such as data type, i.e., $a_c^{(\ell)}$ is either a string or numerical value. We will insert data into the schema that most closely matches the input document image.

To complete our background definitions, we use $\mathcal{H} = \{h_r\}$ to represent a set of human-tunable heuristic functions. We define a particular heuristic function as accepting a visual context window and producing an unnormalized, nonnegative score that tells us how relevant the context-enabled words are to a particular attribute. The exact usage and properties of these will be further explained in Section 3.7.

With this background information, we can formally define our task as follows. Given an input image I, set of schemas $\{S_\ell\}$, and set of heuristic functions $\mathcal{H}$, we wish to create the maximum likelihood assignment of word sequences to attributes. However, it is not necessary that all word sequences will be assigned to some attribute or that all attributes will have some a word sequence assigned to them. This corresponds to having irrelevant text in the image and an attribute not present in the image, respectively.

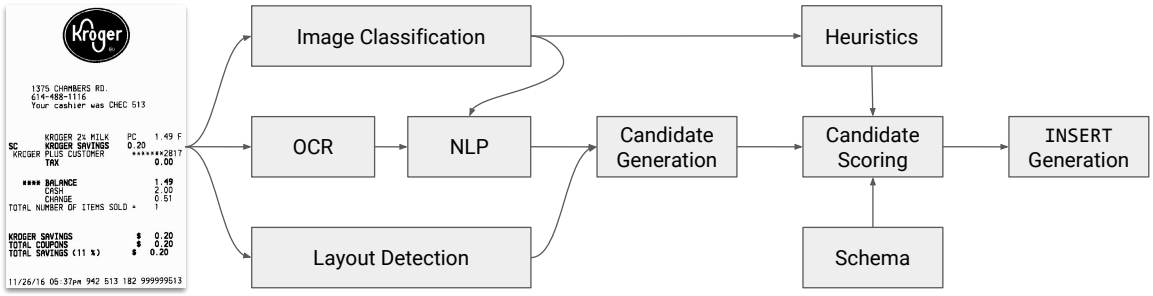

Figure 3.1: An overview of Ifr. We combine computer vision and natural language processing, along with analyst-defined heuristic functions to generate a final `INSERT` statement.

3.2 Overview

An overview of Ifr is shown in Figure 3.1. To start the process of producing structured `INSERT` statements, we are given an input document image and need to determine which schema, and also heuristic functions, to select. For this, we use a state-of-the-art image classification architecture to select the best schema. After we know which schema and heuristic functions to use, we run the image through a state-of-the-art OCR system to produce a bag-of-words with all of the text in this image. From here, we split into two branches that process the image and text independently. Along the text branch, we take the bag-of-words and prune irrelevant words, i.e., *stopwords* and topic-dependent words entered by the analyst based on the schema. To avoid *homoglyphs*, i.e., similar graphemes, we convert all of the text into a *meta-word space* where all homoglyphs map to the same meta-word character in the space. In the image branch, we run a variant of bottom-up segmentation to segment our image into rows and columns. Where these branches merge, each word in the bag-of-words is augmented with its spatial relationships to other words; we call this a

data-box. Additionally, we combine one or more words into the same *data-box* if they semantically or spatially belong together using several heuristics. After creating the list of data-boxes, we use the analyst-entered information and feedback to solve the schema matching problem and assign a data-box to each attribute, if possible. The rest of this chapter describes the components and processes outlined above in more detail.

3.3 Document and Schema Classification

The first step in extracting structured information into a schema is to select the most likely schema. We use a vision model to map the input document image to a schema in the database or schema warehouse. This makes two assumptions: there exists a dataset of document images for each schema in the warehouse, and there is a one-to-one relationship between document images and schemas. For our vision model, we chose, using 5-fold cross validation, an 18-layer ResNet [15] architecture to be our image classifier, where the classes are the schemas. This produces the most likely schema.

However, it is possible for the classifier to select the wrong schema. In this case, we may still receive a partial `INSERT` statement, depending on the similarity of the schemas of the incorrect schema the classifier produces and the true schema of the document.

3.4 Text Analysis

Before starting with text analysis, we need to generate the bag-of-words and bounding-boxes W from the document image; we use a state-of-the-art OCR system[3] to produce a semi-structured W. Afterwards, we process this independently of the image. We call the resulting words and bounding-boxes around them *text-boxes*. This section describes the details of our NLP processes. We outline the various types of pruning and discuss the problem of homoglyphs.

There is no guarantee that the resulting set of text-boxes W is ordered, so we order them using the coordinates of the bounding-boxes. Since we assume English text, we order from left-to-right and top-to-bottom. In a different language, we would order these text-boxes differently. Additionally, we use $before(b)$ and $after(b)$ to denote the prior and subsequent text-box to b. If b is at the beginning of a row, then $before(b)$ refers to the last text-box in the row above, and, if b is at the end of a row, $after(b)$ refers to the first text-box in the subsequent row.

3.4.1 Homoglyphs

The words in the set of text-boxes W may contain homoglyphs, which are detrimental to information extraction. Homoglyphs are similar graphemes, i.e., characters that look similar. For example, "O" (uppercase O), "0" (zero), and "o" (lowercase o) are homoglyphs of each other. Because of the visual similarity of these homoglyphs, OCR system may confuse these characters, i.e., produce the incorrect homoglyph, when they produce the bag-of-words. For instance, the OCR system may produce

[3]Google Cloud Vision. `https://cloud.google.com/vision`

"0lentangy River Road" with a zero instead of "Olentangy River Road" with an uppercase O. In our dataset, we found 10% of the characters produced by OCR to be homoglyphs. Out of these homoglyphs, the most frequent mistakes were confusing "0" (zero) and "O" (uppercase O), "0" (zero) and "o" (lowercase O), and commas and periods. In the case of commas and periods, floating-point numbers, like prices or milligrams, become more difficult to detect.

Our solution is to transform the homoglyph-riddled natural word space into a homoglyph-invariant *meta-word space* through a non-invertible transform. In this meta-word space, all homoglyphs are send to the same meta-word, e.g., "O" (uppercase O), "0" (zero), and "o" (lowercase o) are send to the same meta-word character `#15`. Before executing subsequent algorithms, we convert the input text into its meta-word space representation to neutralize the effect of homoglyphs. Since this is implemented as a simple lookup table, the complexity to translate a word of length λ into the meta-word space is simply $\mathcal{O}(\lambda)$, i.e., each character is translated in $\mathcal{O}(1)$, which is negligible. Through this thesis, we abstract this meta-word space and refer to W to be both the natural word space and meta-word space accordingly.

3.4.2 Text Pruning and Tagging

For a given image, we now possess a set of text-boxes W of all text and their bounding-boxes on the image. However, many of these words are actually *stopwords*, e.g., words like "the", "a", and "an". Keeping these words for subsequent steps leads to more false positives since IFR may assign a score to one of these stopwords, potentially marking it as an `INSERT` value. To remedy this, we can prune stopwords: [3] provides a set of 153 stopwords we use for pruning. Additionally, the complexity

of subsequent algorithms depend on the number of words, therefore, by pruning these stopwords, we improve the runtime of subsequent algorithms.

The stopwords removed by [3] are general stopwords. In context-specific applications, the analyst may want to remove words that are problem-specific. For example, the words "welcome" and "thank you" appear very frequently in the domain of receipts. These are domain-specific words, and IFR allows the analyst to enter these words to be pruned by this step.

However, IFR does not blindly remove stopwords because they may be part of an entity we wish to extract. Therefore, we apply heuristic functions, particularly in lookup and regular expression tables (Section 3.7), before pruning and retain any entities that are detected. This prevents us from removing context or general stopwords that actually belong to an entity.

As an additional step to each text-box, we also run a tagger from [3] and cache the resulting tags for each text-box. For example, we would like to know if a text-box contains a numerical value or not. In the implementation of the heuristic functions, the analyst may use these cached tags in computing the score of the text-box (Section 3.7). When we merge these with visual information to create data-boxes, we combine the tags of the constituent text-boxes (Section 3.6).

3.5 Pixel Analysis

In parallel to processing the image's text, we infer the visual structure by utilizing a modified bottom-up segmentation algorithm. We perform post-processing on our image, such as a median filter, to clean up any noise that may affect the results of the segmentation. This produces visual cells or blocks we call *visual-boxes*.

We use a bottom-up segmentation algorithm, specifically, to make no a-priori assumptions about the visual structure of the image (as opposed to a top-down approach, which requires some knowledge of the input image structure beforehand). Our variant differs from the existing literature on structure detection [12, 20] because much of this literature exploits image elements such as lines, edges, borders, and other metadata and builds an arbitrary segmentation. However, we enforce a tabular structure over the document image because they have an inherent organizational structure meant specifically for human use.

Algorithm 3.5.1 shows our variant of bottom-up segmentation. First, we create horizontal segments by considering the occurrence of black pixels, which denote either the start or end of a section, depending on if a section had already been started. ς is an indicator of the row that started the segment or 0 if a section has not been started yet. After computing these horizontal segments, we look between each non-empty, i.e., contains black pixels, region and perform a similar technique to compute the vertical segments, which, when combined with the bounds of the horizontal segments, become the visual-boxes.

Figure 3.2 illustrates several challenges to segmentation on unstructured document images. This algorithm requires the input image to axis-aligned, since the resulting segmentation is axis-aligned, and mostly free of noise. A rotated or tilted image will produce a poor quality segmentation because the segmentation is axis-aligned. For example, in Figure 3.2 (C), we loose granularity in the horizontal sections. We can de-warp an image using a standard affine transform. If the image has noise, the segmentation will produce more false positives, however, many of these will be removed when we merge the text-boxes and visual-boxes in the next section. For

Algorithm 3.5.1 Bottom-up segmentation algorithm

```
1: function LAYOUT(I)                                          ▷ where I - image
2:     V ← ∅                                                   ▷ visual boxes
3:     H ← ∅                                                   ▷ horizontal sections
4:     ς ← 0                                                   ▷ start of section indicator
5:     for r ∈ I.height do
6:         β ← Σ_j δ(I[r, j], 0)                               ▷ number black pixels in row r
7:         if β > 0 and ς = 0 then
8:             ς ← r
9:         else if β = 0 and ς ≠ 0 then
10:            H.append(⟨ς − 1, r⟩)
11:            ς ← 0
12:        end if
13:    end for
14:    for r_1, r_2 ∈ H do      ▷ iterate over columns between two horizontal sections
15:        if empty(r_1, r_2) then      ▷ consider only regions that contain black pixels
16:            continue
17:        end if
18:        ς ← 0
19:        for c ∈ I.width do
20:            β ← Σ_{i∈[r_1,r_2]} δ(I[i, c], 0)               ▷ number black pixels in column c
21:            if β > 0 and ς = 0 then
22:                ς ← c
23:            else if β = 0 and ς ≠ 0 then
24:                V.append(⟨ς − 1, r_1, c, r_2⟩)              ▷ create visual-box from corners
25:                ς ← 0
26:                c ← c + ω                                   ▷
27:            end if
28:        end for
29:    end for
30:    return V
31: end function
```

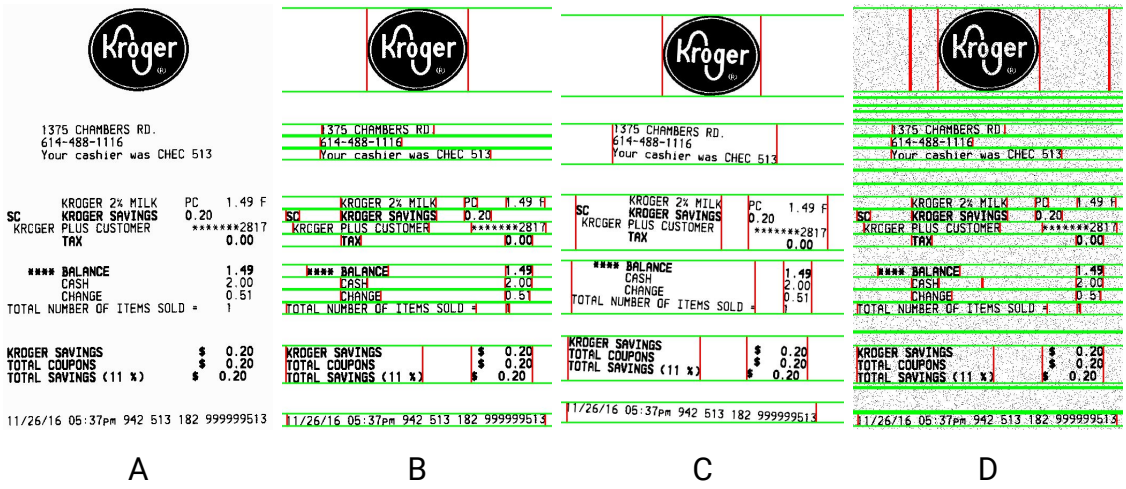

Figure 3.2: Segmentation is affected by changes in input image orientation and noise. (A) The input document image. (B) The detected segmentation on the original input. Green lines indicate horizontal sections and red sections indicate vertical sections. (C) We rotate the original image clockwise by 3° and compute a segmentation. (D) We add salt-and-pepper noise to the original image and compute a segmentation over it.

example, in Figure 3.2 (D), we produce more false horizontal segments below the logo. To reduce the amount of noise in the input image as a result of the capturing device, e.g., scanner or phone camera, we use a median filter to remove the salt-and-pepper noise before computing the visual-boxes. Notice we still produce a similar segmentation, even with noise, in Figure 3.2 (D).

Algorithm 3.5.1 considers all n rows and m columns of an input image I twice: once for horizontal section and again for vertical section. The worst-case complexity is linear in the number of pixels: $\mathcal{O}(nm)$. Reducing the image size can improve the speed of the segmentation.

3.6 Discovering Attribute Assignment Candidates

OCR returns a non-contextualized, single-token result in the form of our text-boxes, and the visual-boxes segment the image into non-overlapping regions. However, we expect the text-boxes to overlap with the visual-boxes. To augment each of these textual tokens with context from their surrounding neighbors, we treat the visual-boxes as a *visual context window*. We call these context-enabled entities *data-boxes*.

Algorithm 3.6.1 Candidate Generation

```
1:  function CANDIDATES(I)                                  ▷ where I - image
2:      W ← bag_of_words(I)
3:      V ← segmentation(I)
4:      D ← ∅
5:      for v ∈ V do
6:          d ← ∅
7:          for w ∈ W do
8:              if w ∩ v ≠ ∅ then
9:                  d.push(w)
10:             end if
11:         end for
12:         if d ≠ ∅ then
13:             d ← sort(d)
14:             D ← D ∪ {d}
15:         end if
16:     end for
17:     return D
18: end function
```

Algorithm 3.6.1 shows how we create data-boxes. For each visual box, we consider all of the text boxes and add them to a data-box if they spatially intersect. Then, we sort all of the words inside a data-box based on their spatial position. Hence, the visual structure of the data-box is enforced by the visual-box, and the contents are enforced by the text-boxes.

Algorithm 3.6.1 scans all of the text-boxes for each visual-boxes. Hence, the worst-case complexity is linear in terms of the product of visual-boxes and text-boxes: $\mathcal{O}(|V| \cdot |W|)$. Experimentally, we found that $|W| \geq |V|$; hence, using our NLP pruning techniques, we can reduce the size of the number of words by an average of 28.87%, which contributes to the efficiency of data-box generation.

3.7 Schema Matching

We consider data-boxes and their containing text-boxes as candidates for `INSERT` generation. The attribute value may be either just a text-box or require a context window. Hence, the goal is to create the *best* one-to-one mapping between data-boxes, or text-boxes, and the schema attributes. In most, nontrivial cases, we may have several data-boxes or text-boxes competing for the same attribute, and the most likely pairing must be computed. To do this, we must score each of these data-boxes and text-boxes for each attribute in the schema.

For this scoring, we use heuristic functions, one for each attribute in the schema, defined by the analyst. We formalize a heuristic function as $h(a, b) : S \times D \cup W \rightarrow [0, \infty)$ for an attribute $a \in S$ where $b \in D \cup W$ is a data-box or text-box. This function returns a score representing the relevancy of the contents of b to a, where larger scores represent more relevancy. Notice h is unbounded above. The only constraint placed on these functions is that a score of 0 must denote complete irrelevancy, e.g., $h(\texttt{city}, \text{computer}) = 0$.

The concrete implementation of these functions is left entirely to the analyst. An example implementation is shown in Figure 3.3. These heuristics are the key to the open-domain application of IFR. For example, in the domain of the medical field,

```
    def is_address(self, box, textboxes):
        if box.dtype is not 'str':
            return 0.

        tokens = word_tokenize(box.text)
        last_tok = tokens[-1].lower()

        ms_tok = self.metaspace.convert(last_tok)
        result = self.conn.execute('SELECT COUNT(*) FROM
    StreetSuffix WHERE MetaspaceData = ?;', (ms_tok,))

        if result.fetchone()[0] == 0:
            return 0.
        if not self.is_int(tokens[0]):
            return 0.

        return 1.
```

Figure 3.3: Example implementation of a heuristic function in Python. We use a lookup table and check the type of the data-box.

the analyst scores "MD" higher as an academic degree. However, in the context of finance, an analyst might score the same information "MD" as a state.

We provide several frameworks and utilities to the analyst to make defining these heuristic functions easier. In particular, we provide simple lookup and regular expression tables, both of which take advantage of our homoglyph-invariant meta-word space.

One use of a lookup table is to select between a finite number of options. For example, for a `city` or `state` attribute, there are only a finite number of options. If the content of a data-box or text-box has an exact match, we can return some score C. Otherwise, we provide edit distance, a common practice in NLP, for the analyst to use. For example, $h(\texttt{city}, \text{NewYorker}) = 0.78$ since only two edits are required to match to "New York" for the `city` attribute. However, $h(\texttt{company}, \text{NewYorker}) = C$ (where C is some large, finite value) since "New Yorker" is the name of a fashion store whose entry exists in the lookup table. These examples illustrate the different contexts the heuristic functions must operate under and the use of the heuristic tables in those varying contexts.

Other attributes, e.g., `telephone`, `date`, and `time`, are easier detected using regular expressions than lookup tables. However, these may change depending on locality: United States phone numbers follow "(###) ### - ####" while French phone numbers follow "## ## ## ## ##". In another example, the time may be displayed as "##:## AM/PM" or in military time "####" with any type of separating characters in between. Hence, there is not one standardized format, but we can consider these using the regular expression tables to validate all of these.

The analyst can define and use these regular expressions to help compute the score in the heuristic function. For convenience, we pre-define some regular expressions, such as date formats, time formats, and phone number formats. For use in the heuristic functions, we also provide more sophisticated metrics such as Levenshtein Distance [25].

We also consider more sophisticated auxiliary heuristic functionality that the analyst may use when designing the heuristic functions.

The data-box "HI" may either be the state of Hawaii or the greeting. However, if the city "Honolulu" precedes "HI", then "HI" is the state rather than the greeting. To incorporate this into IFR, we use n-gram checking to consider the sequences of previous and subsequent words. For example, an address may span several tokens and can only be classified as such with the complete address. We may not score "1375", "Chambers", or "Road" highly as addresses. However, the trigram "1375 Chambers Road" would be scored higher as an address. Instead of checking all possible n-grams, we restrict this to only those values of n determined by the analyst. We call this the *n-gram auxiliary heuristic*. Visual-boxes may provide enough context that the address may be found in one data-box, but this is not guaranteed, which is the purpose for this auxiliary heuristic functionality.

A similar concept to the aforementioned n-gram auxiliary heuristic is the notion of scoring a data-box or text-box higher because of its proximity to another word. For example, the data-box that comes after "cost", "balance", or "total" in a receipt or invoice usually contains the price of the order. In another example, we may have the date and time contained in the same visual-box. Hence, the visual context window is too large, and we need to consider the text-boxes. We can score a date attribute

candidate higher if it is before or after a time attribute candidate. However, we must be careful to check for and disallow circular dependencies.

To better illustrate the use of the heuristic functions, consider the heuristic function for the `date` attribute for the financial domain: $h(\texttt{date}, b)$. This heuristic function returns a higher score for data-boxes and text-boxes that match closely with the date. On an invoice, we may find several dates, so each may be an equally likely candidate for the `date` attribute. However, we can achieve a better matching by pairing it with the `time` attribute. In other words, we want to return an even higher score if the `time` attribute is found nearby. We can easily augment our existing heuristic function to add $h(\texttt{time}, \mathit{before}(b))$ to the final score. This will compute the heuristic function score for the `time` attribute for the box previous to b in the ordering. If the time does not come before box b, then $h(\texttt{time}, \mathit{before}(b)) = 0$ and the value of $h(\texttt{date}, b)$ is unchanged. However, if there is a valid value for time before box b, then $h(\texttt{time}, \mathit{before}(b)) > 0$ and the value of $h(\mathit{date}, b)$ is $h(\texttt{time}, \mathit{before}(b))$ larger. In other word, we score box b even higher than if it simply matched the date. As mentioned before, we disallow circular dependences in the heuristic function, e.g., if $h(\texttt{date}, b)$ was computed using $h(\texttt{time}, b)$, then $h(\texttt{time}, b)$ cannot be computed using $h(\texttt{date}, b)$.

3.8 Insert Generation

Now that we have heuristic functions, we can score each of the data-boxes and containing text-boxes to create a *score matrix* M such that $M_{ba} = h(a, b)$ where $b \in D \cup W$ and $a \in S$. In other words, we evaluate each heuristic function for each data-box and containing text-boxes. Hence, for the score matrix, each row corresponds to all of the heuristic functions being evaluated at a single data-box

or text-box. Similarly, each column corresponds to a particular heuristic function evaluated for all data-boxes and text-boxes. The result is a rectangular matrix M. Since the number of data-boxes and containing text-boxes usually outnumbers the attributes in the schema, we cannot directly use the Hungarian method to solve for the assignments [21]. Additionally, it is not necessary that all attributes be present in an image.

Instead of the Hungarian method, we use the score matrix to create probability distributions over all of the data-boxes and text-boxes for each attribute. To create these, we use a softmax classifier as in Equation 3.1.

$$p(b \mapsto a) = \frac{\exp[h(a, b)]}{\sum_{b' \in D \cup W} \exp[h(a, b')]} \tag{3.1}$$

This computes the probability that a box b should be assigned to attribute a. We apply this softmax function to each column in our score matrix to create a probability distribution for each attribute over all of the data-boxes and text-boxes. Then, to create the assignment, we simply use the maximum likelihood estimate $\operatorname{argmax}_b p(b \mapsto a)$. If an attribute is not present in an image, then it must be the case that $M_{ba} = 0 \; \forall \; b \in D \cup W$. When we apply the softmax function, the resulting distribution will be the uniform distribution. Hence, we can easily detect this and do not create an assignment for these attributes.

Figure 3.4 shows an example assignment of attributes using IFR. Using these assignments, we can create an `INSERT` query.

`Q: INSERT INTO` T $(a_1, a_2, \ldots, a_k)$ `VALUES` $(b_1, b_2, \ldots, b_k)$;

Figure 3.4: An example assignments of attributes to data-boxes. Using the scoring matrix, we can create the maximum likelihood assignment.

Query `Q` inserts a new record into the table with schema S with attributes $a_1, a_2, \ldots, a_k$. The values $b_1, b_2, \ldots, b_k$ are the values of each of the data-boxes and text-boxes.

The score matrix causes the average complexity for generating an assignment to be $\mathcal{O}(|D \cup W| \cdot |S|)$ because we evaluate each heuristic function for each attribute and for each data-box and text-box.

3.9 Limitations

OCR is one of the largest limiting factors of IFR since the later steps are dependent on the accuracy of OCR, which we use as a commodity. Any text in the input image not detected by OCR, such as handwritten poor quality, or occluded text, will not be available for NLP processing, the score matrix, and the final `INSERT` matrix. This phenomenon of error propagation can be observed in our experiments section.

Another limiting factor is the use of heuristic functions entered by the human analyst. If a datum is missed by a heuristic function, it will not be a candidate for `INSERT`. In other words, the resulting quality of IFR is highly tied to the quality of the heuristic functions. However, using heuristic functions, we do not require any training data and can instantly apply changes in rules.

Chapter 4: Experiments

This section evaluates the quality, performance, and usefulness of IFR through an extensive set of experiments. First, we describe our experimental setup (Section 4.1). We compute precision and recall for the different steps of IFR (Section 4.2.1 – 4.2.2). Also, we position IFR against regular-expression-based search, conditional random fields, and related work (Section 4.2.3). An ablation study illustrates the improvement in quality after using the visual context window and NLP pruning (Section 4.2.4). We measure performance of all steps of our approach (Section 4.3). Finally, we conduct a user study comparing IFR quality to human quality (Section 4.4).

4.1 Experimental Setup and Datasets

All experiments were conducted on a Google Cloud Compute Engine instance of 8 vCPUs and 8GB of RAM. IFR itself is written entirely in Python 3.5, and SQLite is used as the database we create `INSERT` statements for. We use the NLTK Python library [3] for pruning and part-of-speech tagging and OpenCV for the bottom-up segmentation algorithm.

For evaluating the quality of IFR and illustrating the open-domain applications, we use three datasets across three different domains: financial, nutrition, and medicine, i.e., the Labcorp dataset. All of the images in our dataset have varying sizes and

resolutions. We also use dataset augmentation to train our classifier and compute performance. Each of the schemas for these datasets were simplified from [34], a public warehouse of schemas.

1. **Financial**. We collected a dataset of 20 receipts and invoice across Columbus, Ohio. These have an average of 56 words and 6 attributes per image. We use the following schema for this dataset: [`date`, `time`, `address`, `city`, `state`, `phone`, `cost`].

2. **Nutrition**. For this nutrition data, we collected 20 nutrition facts labels from various foods. These have an average of 53 words and 6 attributes per image. For this dataset, we use the following schema: [`num_servings`, `serving_size`, `calories`, `cholesterol`, `sodium`, `protein`].

3. **Labcorp**. In addition to these collected datasets, we also use the Labcorp dataset of medical testing documents [22]. These consist of 179 real document images with an average of 256 words and 6 attributes per image. We use the following schema for this dataset: [`specimen_number`, `account_number`, `age`, `dob`, `doc_date`, `doc_time`].

4.2 Quality

For quality, we report precision and recall. However, these metrics have different semantics for different components of Ifr, as shown in Table 4.1. In general, precision is the number of correct results over the number of retrieved results, and recall is the number of correct results over the number of ground-truth results. Note that we do

Component	Precision	Recall
OCR	correctly detected words over all words	correctly detected words over ground-truth words
NLP	-	refined words over ground-truth words
`INSERT`	IFR mappings over all mappings	IFR mappings over ground-truth mappings

Table 4.1: Precision and recall definitions for IFR quality.

not consider precision for the NLP pruning step because the purpose is simply to filter or prune irrelevant words while retaining the relevant words.

4.2.1 Document and Schema Classification

As mentioned in Section 3.3, we use an 18-layer ResNet model. This was trained using the Adam optimizer [19], with the default parameters mentioned in [19]. In addition, we used dataset augmentation that rotated, flipped, and cropped-and-scaled our document image dataset to reduce overfitting and increase the dataset size. We report a test set accuracy of 79.54%.

4.2.2 IFR Quality

Figure 4.1 shows our precision and recall results averaged across the images in our datasets. Notice we observe an *error propagation* phenomenon: `INSERT` quality depends on NLP quality, which depends on OCR quality. Hence, any errors or missing text from OCR propagate through the steps of IFR.

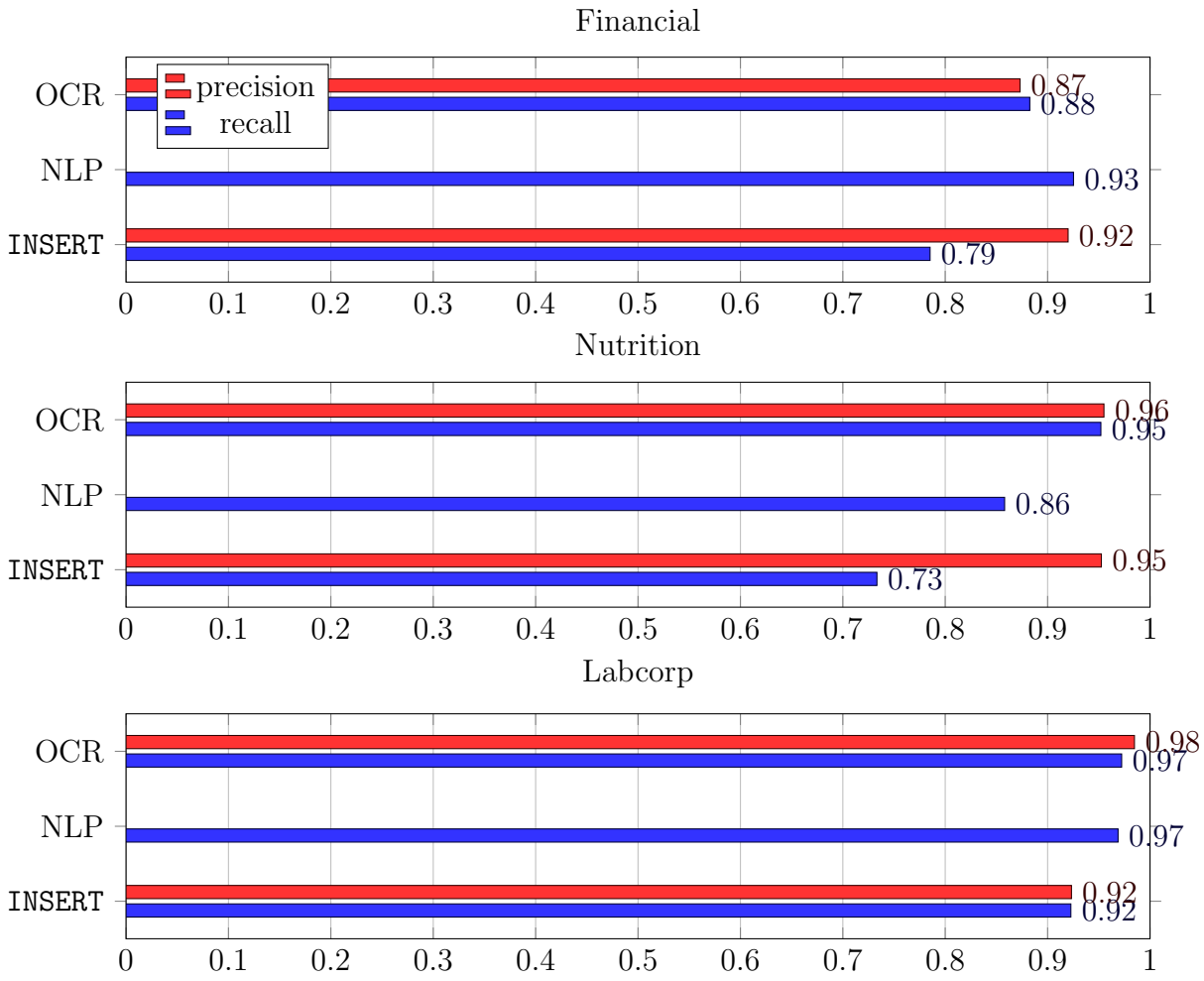

Figure 4.1: IFR quality for each dataset for each component. Notice that we do not compute precision for the NLP pruning step because the purpose of that step is to *prune* irrelevant words and text-boxes.

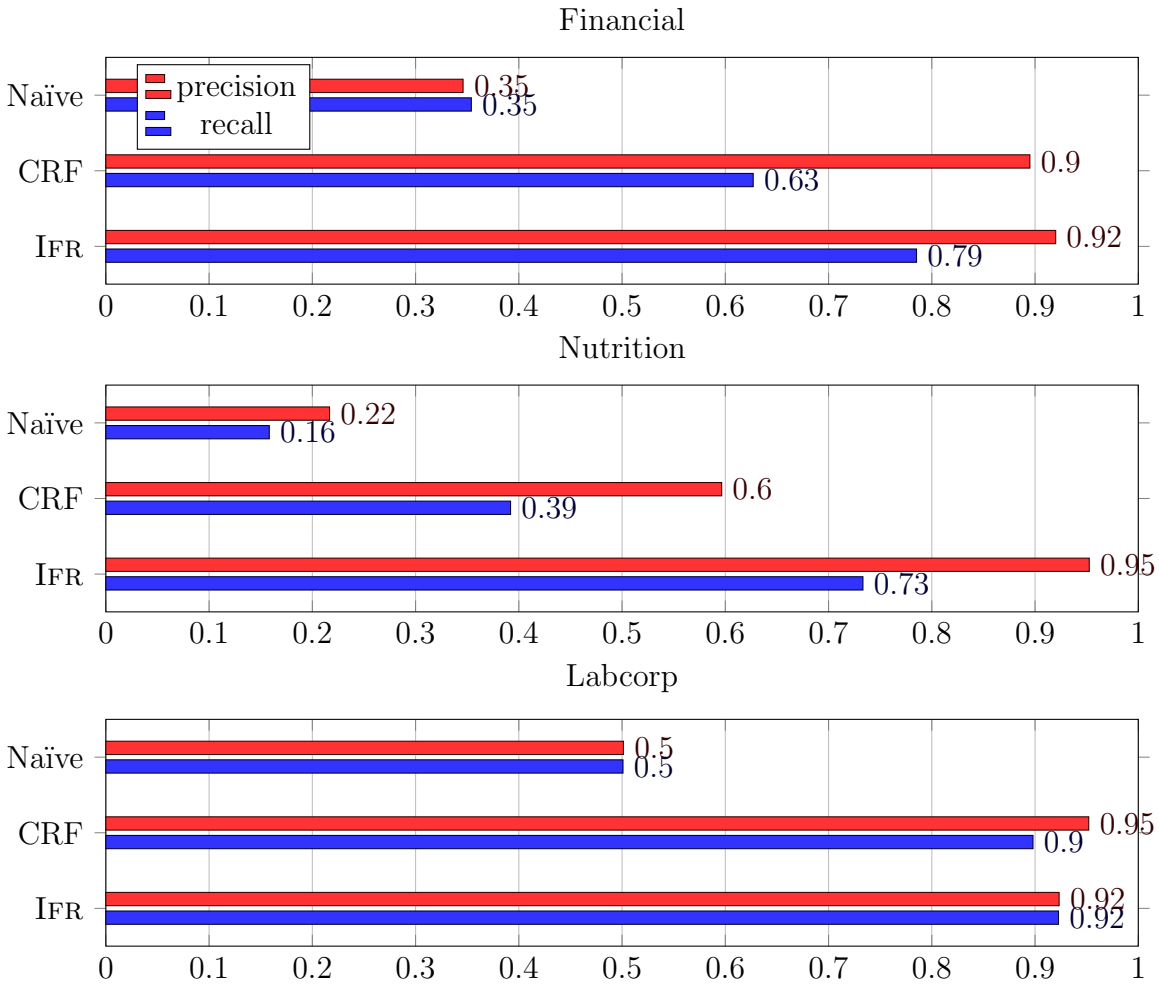

Figure 4.2: Ifr `INSERT` quality compared to the naïve approach and conditional random fields (CRF). The naïve approach uses only regular expressions for schema matching, and the CRF is sequence model learned from our document image corpus. Ifr achieves higher precision and recall than both CRFs and the naïve approach.

4.2.3 Comparison

Figure 4.2 compares Ifr with the naïve approach and conditional random fields. The naïve approach ignores all visual information; we use only regular expressions to parse the resulting OCR text. We attribute the improvement of Ifr to the homoglyph-invariant meta-word space and richness of user-defined heuristic functions combined with knowledge of the visual structure. Regular expressions can be adapted to become homoglyph-invariant only by exhausting all possible combinations of all possible homoglyphs, a combinatorially large problem. Additionally, we may encounter several tokens that a regular expression matches and may not be able to disambiguate between those tokens. However, knowing the visual context and using heuristics, Ifr can disambiguate between these, e.g., we can score higher to the numerical values near "total" or "balance".

In addition to the naïve approach, we also use a more sophisticated sequence model called a conditional random field (CRF) [23]. Traditionally, for NLP tasks, CRFs have been used for named entity recognition and part-of-speech tagging [28, 35]. However, we utilize CRFs to recognize schema attributes using only the OCR text. The input to the CRF is a sequence of text, which is converted into features such as a context window of words around the query word, capitalization, and punctuation. The CRF produces a sequence of labels, in IOB format, where each is an attribute or `NA` for "not an attribute". We group the resulting sequence of labels to create the `INSERT` mapping, e.g., for two consecutive `city` labels, we concatenate their respective words into one entity. We report precision and recall, same as the `INSERT` quality of Ifr.

In both cases, we show that Ifr, on average, outperforms the CRF. However, this comparison is an indirect comparison since it does not consider any visual features

or homoglyphs; IFR considers both. The CRF also requires training data while IFR can function immediately when the heuristic functions are defined. Additionally, IFR uses schema-level information while the CRF does not.

We also compare the quality of IFR to Shreddr [7], a related work that uses image templates for information extraction. We performed an apples-to-oranges comparison against Shreddr's image template approach and IFR and observed very low values for precision and recall on the financial and nutrition datasets. We report 0.0125 precision and 0.0100 recall for the financial dataset and 0.0417 precision and 0.0417 recall for the nutrition dataset. We attribute these low values to the varying visual structure of the images in those datasets. However, Shreddr performs significantly better on the Labcorp dataset, with 0.8172 precision and 0.7952 recall, because many of these forms follow the same visual format. Notice that IFR still produces an improvement in both precision and recall over Shreddr.

4.2.4 Ablative Analysis

We conducted an ablation study to pinpoint the contribution of each component of IFR. We remove any NLP pruning, both stopwords and context-based words, and throw away any additional textual information, e.g., data type, and compute precision and recall for the resulting `INSERT` statement. When we omit visual information, we do not use any visual context window or spatial relationships betweens text-boxes.

Figure 4.3 shows the results of our ablation study. We observe higher precision, recall, or both using both NLP and visual information. For the financial dataset, both precision and recall increase because there is more visual variation in the input images than the other two datasets. This is also true for our nutrition dataset. We notice

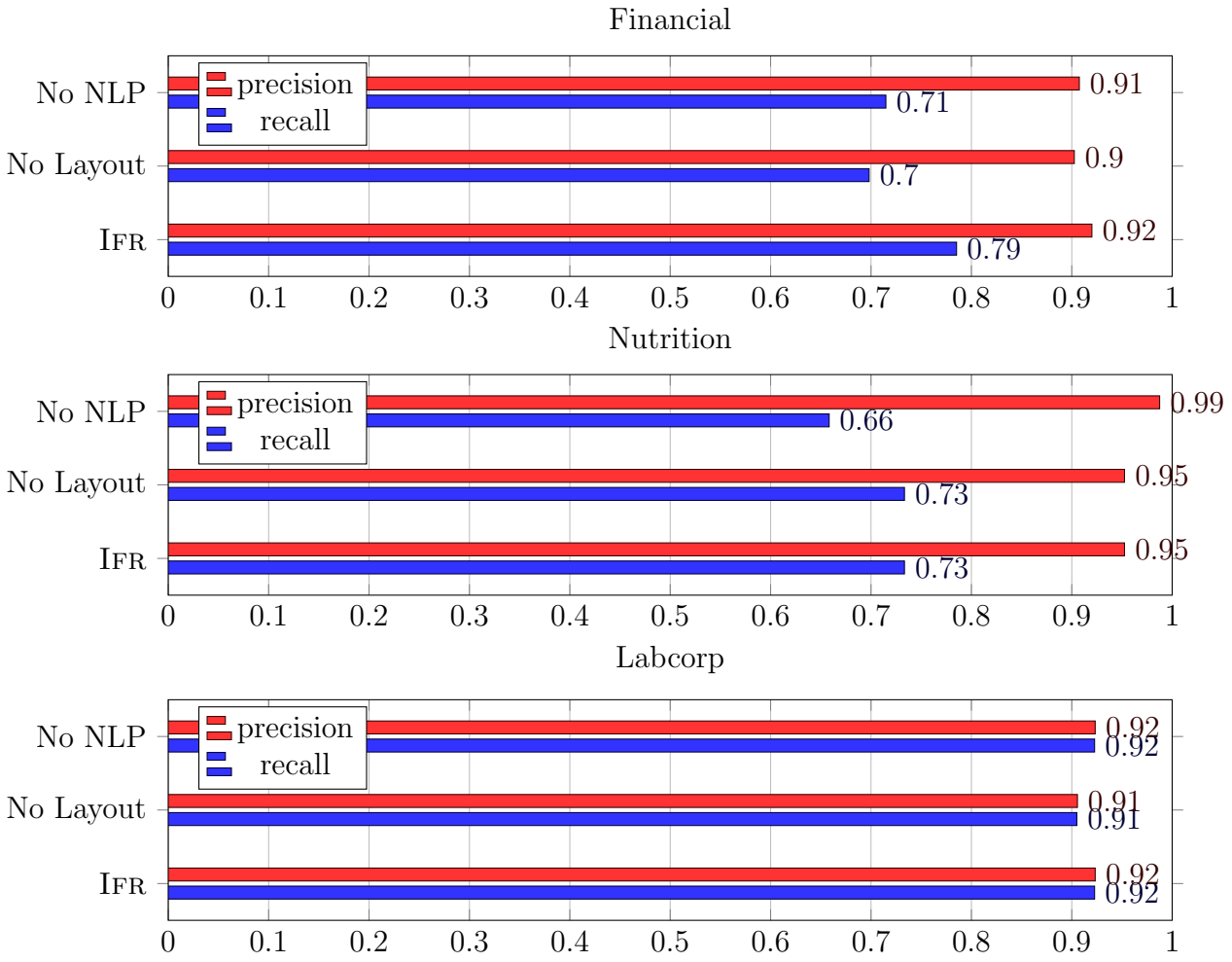

Figure 4.3: Ablation study. We consider the effects of forgoing any text processing and omitting visual information on the final quality of the `INSERT` statement. Keeping text and visual information allows us to achieve a higher quality for our datasets, on average.

that removing layout does not affect the end `INSERT` statement quality because the images in the nutrition dataset are more visually uniform than the financial dataset. Finally, since the images in the Labcorp dataset are also fairly standardized, the final `INSERT` quality is more consistent, similar to the nutrition dataset. Hence, the visual context window from the segmentation algorithm improves quality as the input document images become more unstructured and ad hoc.

4.3 Performance

In addition to high quality, Ifr needs to be fast for the analyst to iterate on heuristic functions. We conducted a performance study that breaks down the execution time of Ifr into the following components: retrieving the bag of words from OCR (OCR), text processing (NLP), visual segmentation and candidate generation, constructing the scoring matrix and attribute assignments, and generating and executing the `INSERT` statement.

Figure 4.4 shows the breakdown, averaged across all images in our dataset with dataset augmentation ($15,000$ images across all datasets). We observe that OCR requires most of the time, 1.4s including network latency. However, other factors, such as image size and resolution, affect this OCR time. We use Google Cloud Vision, which requires a web request to be transmitted. We could improve performance by using an on-device OCR engine, such as Tesseract OCR; we found that using this is 424ms faster on average. However, we lose 0.1620 precision and 0.1043 recall in the final `INSERT` statement, averaged across all of our datasets.

If we exclude OCR, then segmentation and candidate generation require the most amount of time. Segmentation (39.1ms) requires more time than candidate generation

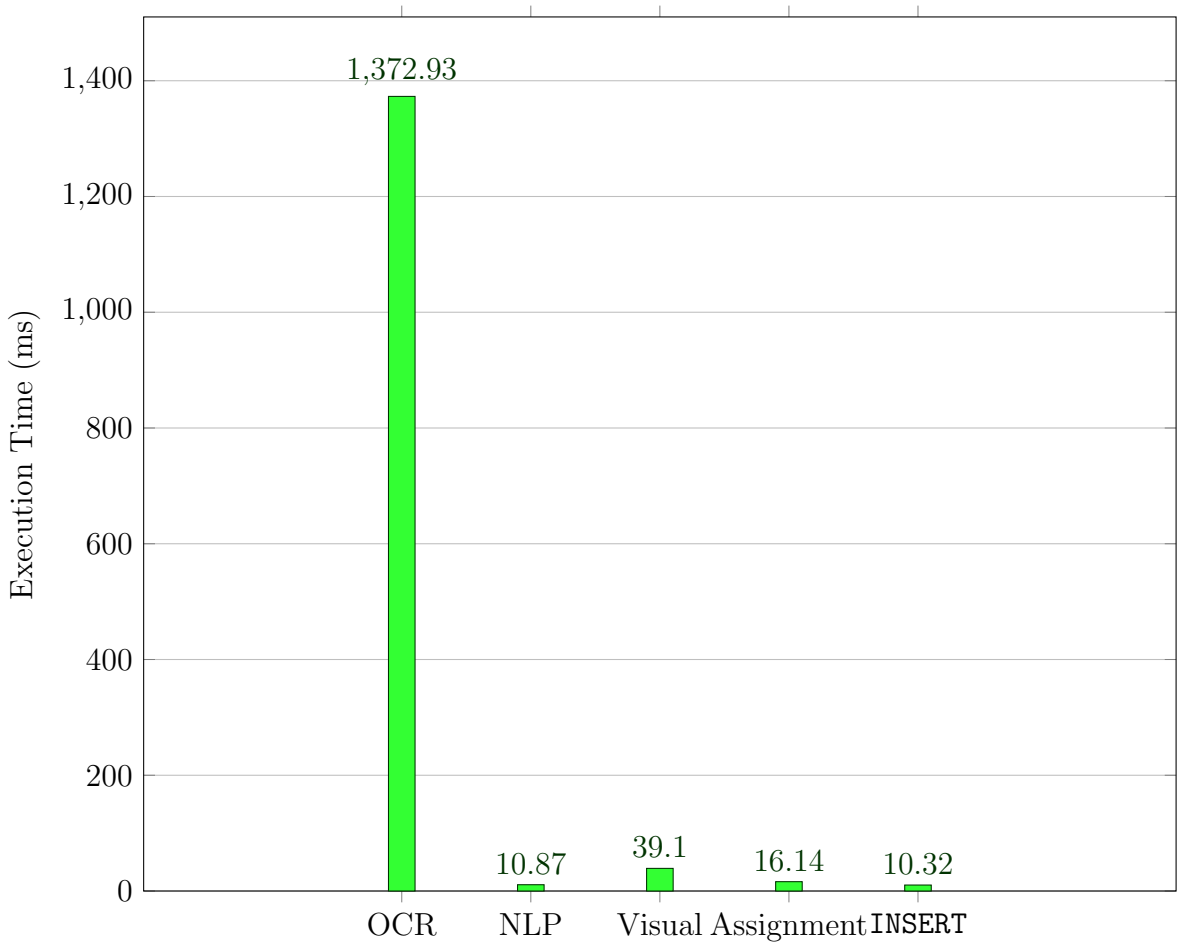

Figure 4.4: IFR performance. OCR, the commodity, requires more time than all other components by two orders of magnitude.

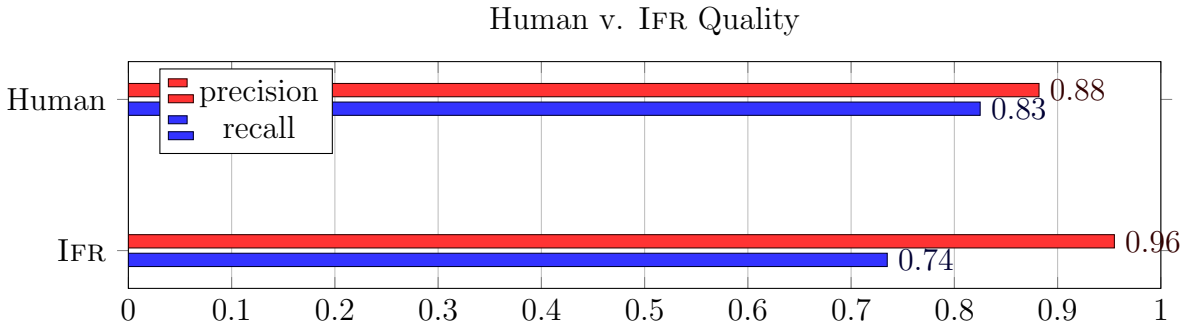

Figure 4.5: Comparison of human quality to IFR quality on the financial dataset. IFR produces higher precision assignments than humans.

because it must iterate through all pixels in an image. Candidate generation requires much less time because it iterates through the number of data-boxes and text-boxes and attributes. In practice, we observe the number of attributes is much smaller compared to the number of data-boxes and text-boxes. One performance improvement is to downsample the image, reducing the total number pixels the segmentation algorithm has to process. Finally, the NLP processes and `INSERT` generation and execution take the least amount of time.

Since the text and visual processing branches operate independently, we run them in parallel. While we retrieve the bag-of-words and bounding boxes from the OCR engine, we are visually segmenting the image. Hence, this parallelism causes IFR to be bounded only by the slowest component.

4.4 User Study

We conducted a user study of 100 Mechanical Turk workers. We asked these workers to manually digitize input images in the financial dataset by annotating the

schema attributes. From their results, we computed the same precision and recall metrics as the final `INSERT` statement.

Figure 4.5 shows the results of our Mechanical Turk user study. IFR can better identify the correct value of a field than the human workers. However, they achieve a higher recall than IFR. This is because workers missed fields that were not in regular places and gave up after they could not find them. For example, we noticed that in receipts where the state was near the tax field than the city, workers tended to miss this and omit that attribute entirely. However, IFR correctly detects this because it considers all text in an image when computing the score matrix and assignments instead of an attention-based approach.

Chapter 5: Conclusion

In this thesis, we have shown and evaluated IFR, an approach to extract schema-ready information from unstructured document images. We evaluated the quality of our approach across three datasets, financial, nutrition, and Labcorp, to show that IFR produces high precision and recall scores overall. We also compare with the naïve regular expression search and conditional random fields. Our approach is parallelized and runs entirely in a few seconds per document image, including network latency. To compare our approach humans, we conducted a Mechanical Turk study where participants were asked to digitize several document images. Our results showed that our approach was on-par with human-level performance.

5.1 Contributions

We review the primary contributions of our approach as follows.

- We adapted standard bottom-up segmentation to create a document-image-specific segmentation, regardless of the visual structure (Section 3.5).
- We illustrated an approach to merging visual and textual information from the document image with schema information from the database to create `INSERT` candidates (Section 3.4–3.6).

- Finally, we described an analyst-defined heuristic approach to scoring these candidates to create the attribute assignment and subsequent `INSERT` statement (Section 3.7–3.8).

5.2 Future Work

We describe two directions of future work: mixing machine learning and human-readable heuristics and a stronger document classification model.

5.2.1 Learned Schema Matching

Currently, IFR uses no machine learning from prior examples in the database to help in information extraction. We choose not to convert the user-defined heuristic functions into a deep learning approach since they are maintained by the engineer and meant to be human-readable. Therefore there is a need to investigate techniques that combine visual and textual machine learning models with human-written heuristics. DeepDive [32] addresses this using statistical inference and learning on a factor graph where the variables represent human-understandable relations.

5.2.2 Document Classification

Currently, we used a state-of-the-art ResNet [15] model trained on 3 different document classes. However, the content of the document is as important as the visual structure. Instead of using just visual features for document classification, one direction of future work is to incorporate textual information as well. This can be accomplished by passing the document image through a vision model, much like what we do now, and passing the OCR text of the document image through a language

model augmented with character-level features, such as an LSTM [16] or character-aware LSTM [18]. The result is two vectors: one representing the visual features of the image and another representing the textual features. These can then be combined using the Hadamard product, i.e., element-wise multiplication, which tends to perform better than concatenation or addition. After this fusion, the result can be fed through a deep neural network to classify the document.

Bibliography

[1] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *ACM SIGMOD Record*, pages 54–59, 1999.

[2] P. A. Bernstein, J. Madhavan, and E. Rahm. Generic schema matching, ten years later. *VLDB Endowment*, pages 695–701, 2011.

[3] S. Bird and E. Loper. Nltk: the natural language toolkit. page 31, 2004.

[4] M. J. Cafarella et al. Webtables: Exploring the power of tables on the web. *VLDB*, pages 538–549, 2008.

[5] J. Chen, A. Abouzied, D. Hutchful, J. Ming, and I. Ghosh. printr: Exploring the potential of paper-based tools in low-resource settings. In *Proceedings of the Eighth International Conference on Information and Communication Technologies and Development*, page 23. ACM, 2016.

[6] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh. Usher: Improving data quality with dynamic forms. *IEEE Transactions on Knowledge and Data Engineering*, 23(8):1138–1153, 2011.

[7] K. Chen et al. Shreddr: pipelined paper digitization for low-resource organizations. *ACM DEV*, page 3, 2012.

[8] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. *WWW*, pages 232–241, 2002.

[9] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *TRC: Emerging Technologies*, pages 271–288, 1998.

[10] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD*, pages 509–520, 2001.

[11] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. *ACM WWW*, pages 662–673, 2002.

[12] K.-S. Fu and J. Mui. A survey on image segmentation. *Pattern Recognition*, pages 3–16, 1981.

[13] M. Ganis et al. Neural network-based systems for handprint ocr applications. *IEEE ToIP*, pages 1097–1112, 1998.

[14] R. H. Gustin, T. W. Livingston, and N. Park. Automated banking system for dispensing money orders, wire transfer and bill payment, 2000. US Patent 6,012,048.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE CVPR*, pages 770–778, 2016.

[16] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Y. Ishitani. Logical structure analysis of document images based on emergent computation. *ICDAR*, pages 189–192, 1999.

[18] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. pages 2741–2749, 2016.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[20] K. Kise, A. Sato, and M. Iwata. Segmentation of page images using the area voronoi diagram. *Computer Vision and Image Understanding*, pages 370–382, 1998.

[21] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, pages 83–97, 1955.

[22] Labcorp. `https://labcorp.com/`.

[23] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML*, pages 282–289, 2001.

[24] Y. Lee, M. Sayyadian, A. Doan, and A. S. Rosenthal. etuner: tuning schema matching software using synthetic scenarios. *VLDB*, pages 97–122, 2007.

[25] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. pages 707–710, 1966.

[26] J. Li, J. Tang, Y. Li, and Q. Luo. Rimom: A dynamic multistrategy ontology alignment framework. *IEEE KDE*, pages 1218–1232, 2009.

[27] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. *VLDB*, pages 49–58, 2001.

[28] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. *ACL CoNLL*, pages 188–191, 2003.

[29] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. *IEEE ICDE*, pages 117–128, 2002.

[30] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. *VLDB*, pages 77–88, 2000.

[31] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. *Fusion*, 1999.

[32] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *VLDS*, pages 25–28, 2012.

[33] D. Niyogi and S. N. Srihari. Knowledge-based derivation of document logical structure. *ICDAR*, pages 472–475, 1995.

[34] Schema.org. `http://schema.org/`.

[35] B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. *ACL NLPBA*, pages 104–107, 2004.

[36] Tabula. `http://www.tabula.technology/`.

[37] A. Tengli, Y. Yang, and N. L. Ma. Learning table extraction from examples. *COLING*, page 987, 2004.

[38] J. Wang, Y. Wang, and Y. Wang. Capff: A context-aware assistant for paper form filling. *THMS*, page 3, 2016.

[39] M. L. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. A unified approach for schema matching, coreference and canonicalization. *ACM SIGKDD*, pages 722–730, 2008.

[40] Worldwide digital image forecast. `http://www.marketresearch.com/IDC-v2477/Worldwide-Digital-Image-Forecast-Capture-8773261/`.